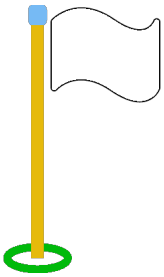
Trabajo Práctico 3
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación – 2° cuat. 2017

Fecha de entrega: 9 de Noviembre de 2017



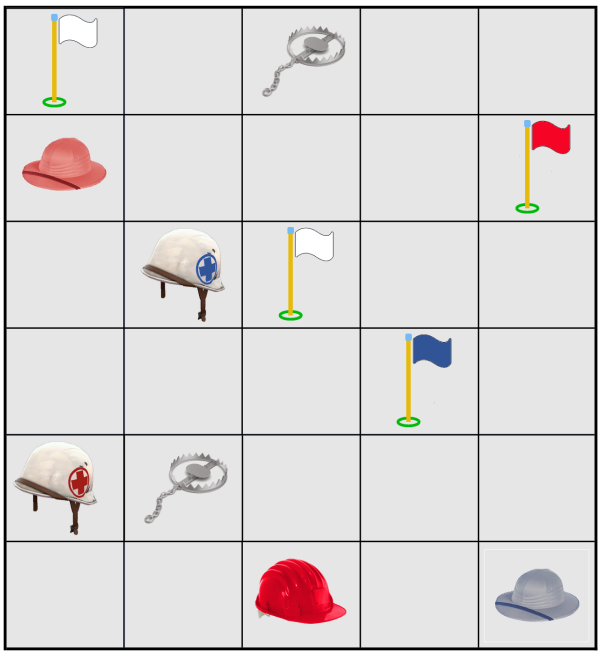
Objetivo

El juego se llama Captura la bandera[1], un juego de estrategia que consiste en conseguir la mayor cantidad de banderas dentro del terreno de juego. Los participantes se dividen en equipos compuestos por personajes con diferentes habilidades y de acuerdo a su formación dependerá su estrategia.

Elementos

- Terreno: matriz en la que se desarrolla el juego. El terreno tiene instaladas banderas y trampas.
- Bandera: las banderas están distribuidas arbitrariamente en el terreno y no tienen equipo asignado al inicio del juego.
- Trampa: las trampas se encuentran ocultas en el terreno y no son visibles para la mayoría de los personajes. Si un personaje pisa una trampa queda inhabilitado y la trampa desaparece del juego.
- Tipos de personajes:
  - Exploradores: son los encargados de marcar (asignar a su propio equipo) las banderas del terreno.
  - Ingenieros: son los únicos que pueden ver las trampas y desactivarlas.
  - Médicos: son los habilitados para rescatar heridos (personajes inhabilitados).

[1] Diferentes versiones con una única bandera: https://es.wikihow.com/jugar-a-capturar-la-bandera

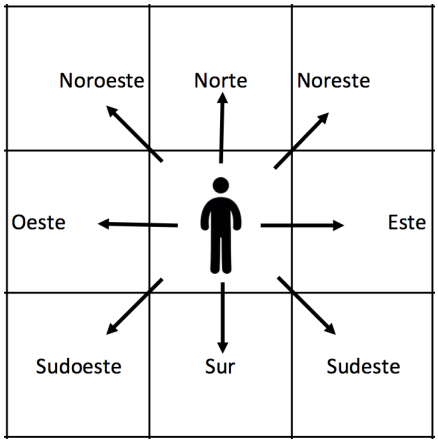


Como se puede ver en la grilla, las banderas pueden o no pertenecer a alguno de los equipos jugadores.

Desarrollo

Los equipos comienzan registrándose en el juego y, una vez que el mismo comienza, cada equipo puede mover un único jugador en su turno. En cada turno, se le indica a un personaje una cantidad de pasos y una dirección para que se mueva. Si el mismo resulta herido, se choca con otro elemento del juego o llega al límite del terreno se detiene. Ningún personaje puede pasar por una casilla ocupada por otro personaje ni por una bandera.

En cada paso que da el personaje, el mismo puede interactuar con los elementos de sus casillas adyacentes:



El Juego

Al iniciar el juego, deben definirse las siguientes características:

- Las dimensiones del terreno
- Las ubicaciones de las banderas
- Las ubicaciones de las trampas

Una vez creado el juego, se debe permitir registrar equipos indicando el nombre, la composición del equipo y las posiciones de cada personaje. El orden de los turnos está establecido de acuerdo al orden en el que se registraron los equipos. No hay un fin de juego establecido, sino que en un paso determinado se identifica como equipo/s ganador/es a los que tengan mayor cantidad de banderas marcadas. Para conocer dicha información, se le puede pedir al juego el conjunto de equipos ganadores. En caso de que no haya ninguna bandera marcada, el conjunto resultante será vacío.

Ejercicio 1. Implementar las clases y métodos necesarios para poder crear banderas y trampas. Las banderas deben ser visibles pero las trampas no. A diferencia del resto de los elementos del terreno, la trampa debe devolver Nil cuando se le pida que se evalúe. A una bandera se le debe permitir cambiar el equipo que la marcó.

Al finalizar este ejercicio deben poder pasar los tests de la clase `TestsEx01`.

Ejercicio 2. Implementar las clases y métodos necesarios para poder crear el terreno de juego. Este se crea con sus dimensiones y debe poder verificarse si un punto cualquiera pertenece al terreno. Además, deben poder agregarse y/o removerse elementos del terreno. Por otro lado, se deben poder seleccionar los elementos del terreno de acuerdo a alguna condición y, dado un punto, se debe poder conocer su contenido mediante los mensajes `at:` y `visibleAt:`, que devuelven el elemento (resp.el elemento visible) en dicho punto, si lo hay, y `nil` en caso contrario.

Al finalizar este ejercicio deben poder pasar los tests de la clase `TestsEx02`.

Ejercicio 3. Implementar las clases y métodos necesarios para poder crear los personajes del juego. Los mismos deben poder responder a los siguientes pedidos:

- a qué equipo pertenecen.
- si tienen la habilidad de sanar.

Dado un personaje, el terreno debe poder responder qué elementos ese personaje puede ver a su alrededor. Además, dichos elementos deben poder ser seleccionados de acuerdo a alguna condición.

Al finalizar este ejercicio deben poder pasar los tests de la clase `TestsEx03`.

Sugerencia: usar los mensajes `allSubclasses` o `eightNeighbors`, `select:` y `collect:` cuando sea conveniente.

Ejercicio 4. Implementar las clases y métodos necesarios para que cada personaje sepa interactuar con su entorno de acuerdo a sus habilidades. Estos deben poder responder a los siguientes mensajes:

- si pueden moverse (es decir, si no están heridos).
- si pueden moverse en una dirección específica.
- inhabilitarse.

- su ubicación.
- si pueden curar.
- ser sanados por otro personaje (esto debe arrojar una excepción si el otro personaje no puede curar).

Además, cada personaje debe interactuar con su entorno y ejecutar la acción correspondiente a su misión en el juego:

- La misión de los exploradores son las banderas. Si uno de ellos pasa por una casilla adyacente a una bandera, la marca con el nombre de su equipo por más de que la misma ya haya sido marcada por otro.
- La misión de los ingenieros son las trampas. Si uno de ellos pasa por una casilla adyacente a una trampa, la desactiva y esta desaparece.
- La misión de los médicos son los heridos (médicos, ingenieros o exploradores inhabilitados). Si uno de ellos pasa por una casilla adyacente a un herido de su equipo, lo sana. Los médicos inhabilitados no pueden sanarse a sí mismos.

Sugerencia: reutilizar los mensajes que permiten conocer el entorno de cada personaje.

Como ayuda, se provee el código del método que implementa el mensaje `moveTo:`, para ser pegado en la clase correspondiente.

```
moveTo: aDirection
    | content |
    (self canMoveTo: aDirection) ifFalse: [ ^self].
    terrain remove: self.
    location := aDirection nextFrom: location.
    content := terrain at: location.
    terrain at: location put: self.
    content ifNil: [self interactWithAllElements] ifNotNil: [ self disable ].
    ^self
```

Notar que la interacción con los elementos adyacentes se realiza siempre que el receptor no haya caído en una trampa.

Al finalizar este ejercicio deben poder pasar los tests de la clase `TestsEx04`.

Ejercicio 5. Implementar las clases y métodos necesarios para que se puedan crear equipos, registrar equipos, jugar y conocer a los equipos ganadores.

La cantidad de jugadores por equipo puede diferir, al igual que su composición.

En cada turno, el equipo correspondiente puede mover un jugador una cantidad arbitraria de pasos en una dirección - siempre que sea posible - luego de lo cual el turno pasa al equipo siguiente.

Al finalizar este ejercicio deben poder pasar los tests de la clase `TestsEx05`.

Ayuda: se recomienda usar los mensajes `groupedBy:` y `detectMax:` para identificar a los ganadores.

Pautas de entrega

El entregable debe contener:

- un archivo `.st` con todas las clases implementadas
- versión impresa del código, comentado adecuadamente (puede ser el propio .st sin los tests)
- **No** hace falta entregar un informe sobre el trabajo.

Se espera que el diseño presentado tenga en cuenta los siguientes factores:

- definición adecuada de clases y subclases (si corresponde), con responsabilidades bien distribuidas
- uso de polimorfismo y/o delegación para evitar exceso de condicionales (no usar más de un **ifTrue:**, **ifFalse:** o **ifTrue:ifFalse:**; sí se pueden usar libremente mensajes como **ifNil:**,**ifEmpty:**,**ifNotEmpty:**,**detect:ifNone:**,**detect:ifFound:**,**whileTrue:**, etc.)
- intento de evitar código repetido utilizando las abstracciones que correspondan.

Consulten todo lo que sea necesario.

Consejos y sugerencias generales

- Lean al menos el primer capítulo de *Pharo by example*, en donde se hace una presentación del entorno de desarrollo.
- Explorar la imagen de Pharo suele ser la mejor forma de encontrar lo que uno quiere hacer. En particular tengan en cuenta el buscador (`shift+enter`) para ubicar tanto métodos como clases.
- No se pueden modificar los test entregados, si los hubiere, aunque los instamos a definir todos los tests propios que crean convenientes.

Importación y exportación de paquetes

En Pharo se puede importar un paquete arrastrando el archivo del paquete hacia el intérprete y seleccionando la opción "`FileIn entire file`". Otra forma de hacerlo es desde el "`File Browser`" (botón derecho en el intérprete > `Tools` > `File Browser`, buscar el directorio, botón derecho en el nombre del archivo y elegir "`FileIn entire file`").

Para exportar un paquete, abrir el "`System Browser`", seleccionar el paquete deseado en el primer panel, hacer click con el botón derecho y elegir la opción "`FileOut`". El paquete exportado se guardará en el directorio `Contents/Resources` de la instalación de Pharo (o en donde esté la imágen actualmente en uso).